CA4006
Concurrent and Distributed Programming
Assessment N°2:

Martin FERRAND - 22104615

Clovis GILLES - 22103376

1. Link to the video presentation:

https://youtu.be/Nxiz9W9k8ys

2. How to compile and run our code:

Before running our Python code, you will need first to have the Rabbit MQ docker running with the following command:

docker run -it --rm --name rabbitmq -p 5672:5672 -p 15672:15672 rabbitmq:3.11-management

You will need Python3 so ensure that it is installed and that pip is installed with it.

To install the required packages please use the following command:

pip install -r .\requirements.txt

The command will just install the pika package (1.3.1).

To run the program, you just need to run the main.py script with Python.

py main.py

or:

python3 main.py

Depending on the installation of python.

3. Design documentation

The main design idea uses the *ComEntity* class which is responsible for all the communications between the funding agencies, the university and the researchers using RabbitMQ.

Each *ComEntity* is a thread and only communicate which other parts of the program using RabbitMQ after being started. The *ComEntity* is an abstract class used to implement the ability to send and receive messages (with the *sendMsg* and the *receiveResponse* methods). And by ensuring that each child can implement a behaviour by overriding the *behaviour* method. The *ComEntity* class also provide minor utility functions.

We have a *TimeKeeper* class that extends *ComEntity* useful to count the time. It sends time update via RabbitMQ to the funding agencies and the universities.

After that we have the funding agency (*FundingAgency* class) that extends *ComEntity* that will receive via RabbitMQ the research proposals of the researchers and will decide if it is

accepted or not based on the amount of money it has left and other criteria.
If the proposal is accepted the researcher will be notified and the university will also be notified.

The *university* class that extends *ComEntity* will keep track of the money the researcher has on the fund that have been allocated when a research proposal has been accepted. The university can deal with the researcher withdrawing money from their account, adding and removing member to their team and managing new research account and the ability to delete them if needs be.

And finally, the *researcher* class that extends *ComEntity* will make research proposal until one is accepted to a funding agency and if the researcher has an accepted proposal they will now work on it (meaning withdrawing money from their account from time to time).

The *research* class is used to manage in a tidy way the interactions between the university and the research accounts.

Since we use RabbitMQ there is no concurrency issues since the threads do not interact directly with each other's.

Our program has some trouble dealing with crashes or start/stop cycles without cleaning the messages stored in RabbitMQ.

4. What is implemented

The following is implemented:

We managed to implement 3 researchers (can be more) dealing with 2 funding agencies (can be more) and one university.

The research proposals are made of an ID (the researcher's ID), a title, a project description and the requested amount of funding.

The funding agency has a budget, a minimum and a maximum for the application fund amount to be accepted and it keeps track of how much it has left.

The funding agency will respond to the researcher if the proposal is accepted or not and will transmit the relevant information to the university if the research proposal is accepted.

The university does store all information related to research accounts, users, and the transactions made and delete the account if the deadline is crossed or if there is no more money on the account.

The researcher who gets a project approved can add and remove researchers. Withdraw funds if there is still money in the account.

The following is not implemented:

The researchers cannot access the details of their account or list the transactions made on that account.

5. Personal reflection

Clovis Gilles:
This assignment allowed me to learn how to use RabbitMQ but not much else since I have already worked with Python during an internship with concurrency and multithreading notions. However I had to learn again the syntax of Python for example ternaries. The TCP protocol inspired me during the creation of our protocol because every communication in TCP implies a response. Therefore our system will require an acknowledgment for each message sent using what we have seen during the lectures.

During this project I have worked on the following python files:

- University.py
- FundingAgency.py
- Researcher.py
- Research.py
- ResearcherAccount.py

And I have worked on the following problems:

- Communication protocol used for each exchanges
- The generation of subject, names and object of the researches

If I have to rework or redo this project I would put more attention to the communication protocol and put more time and effort in the behaviour of the researchers. I would have liked to have multiple universities. I would also have liked to give an outcome to the research with maybe a success chance and some consequences.

Martin Ferrand:
To me the challenge of this project was to work with RabbitMQ which I did not know. I used the lecture and the online documentation of RabbitMQ. Python was not a problem to work with. I was familiar with every other notions used in our code.

I worked on the following python files:

- TimeKeeper.py
- ComEntity.py
- main.py

I have worked on the following problems:

- How to use RabbitMQ.

- Creation of the report

Next time I will try to make a more resilient system. They program work quite well if no issue arise but restart of the application often lead to issue with the messages the RabbitMQ server has left from the previous session.